Many thanks to **Eben Roux** for code documentation and review.

Chapter 3 : The Particular Service Platform

In this chapter, we will be focusing on the Particular Service Platform that includes ServicePulse, ServiceControl, ServiceInsight and ServiceMatrix.

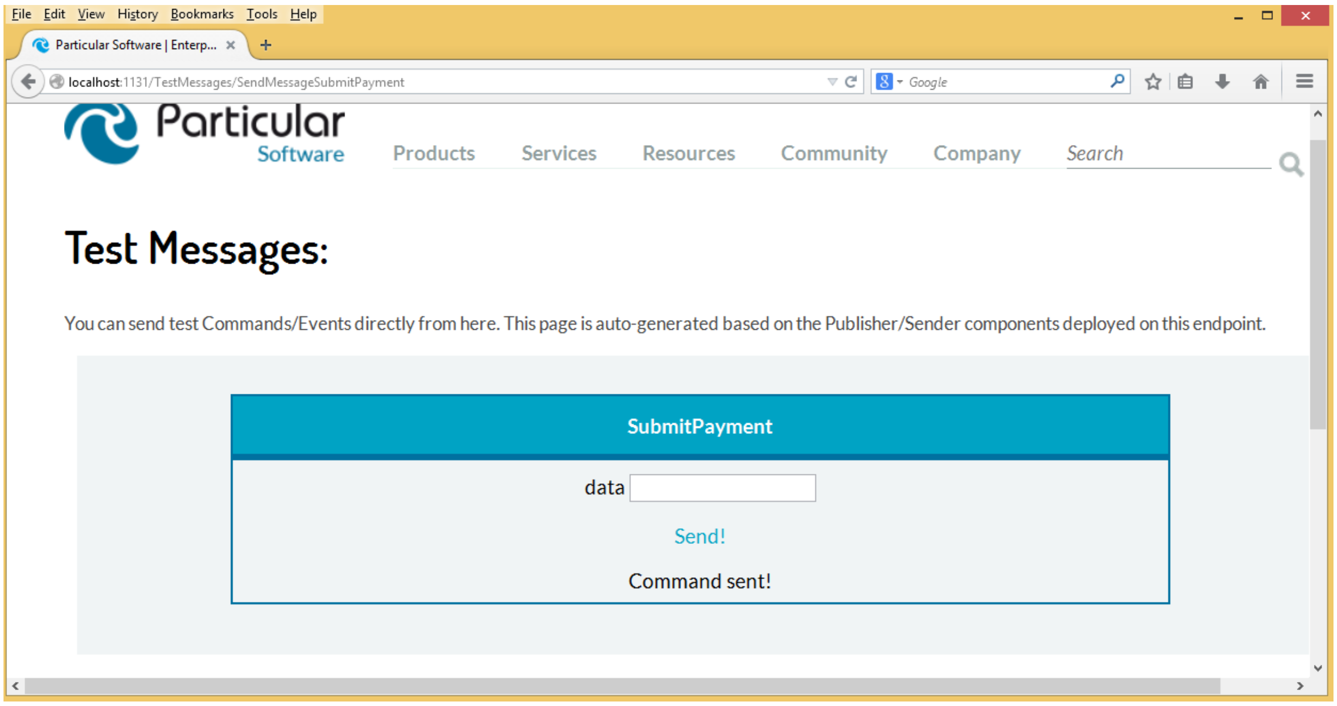
As the names imply ServicePulse gives us a pulse on the messages, services and endpoints. ServiceControl is the control API that ServicePulse and ServiceInsight depends on in getting its internal information. ServiceInsight gives us graphical and message level drilldown into the services, endpoints and messages that also includes a saga drilldown.

ServiceMatrix is the graphical interface into code generation for NServiceBus endpoints, services, and messages in a Visual Studio canvas.

The Source code in this section:

In this section the NServiceBus Service platform must be installed to include ServiceMatrix, ServicePulse and ServiceInsight.

In this section, we will be using the **PaymentEngine-Start** solution created with the ServiceMatrix, so ServiceMatrix must be installed. This will be the ServiceMatrix walkthrough. This will be a request-repsonse solution built with ServiceMatrix. The outcome of a sent payment will look as follows in a Payment:


File Edit View History Bookmarks Tools Help
Particular Software | Enterp...
localhost:1131/TestMessages/SendMessageSubmitPayment
Google
Particular Software
Products
Services
Resources
Community
Company
Search

Test Messages:

You can send test Commands/Events directly from here. This page is auto-generated based on the Publisher/Sender components deployed on this endpoint.

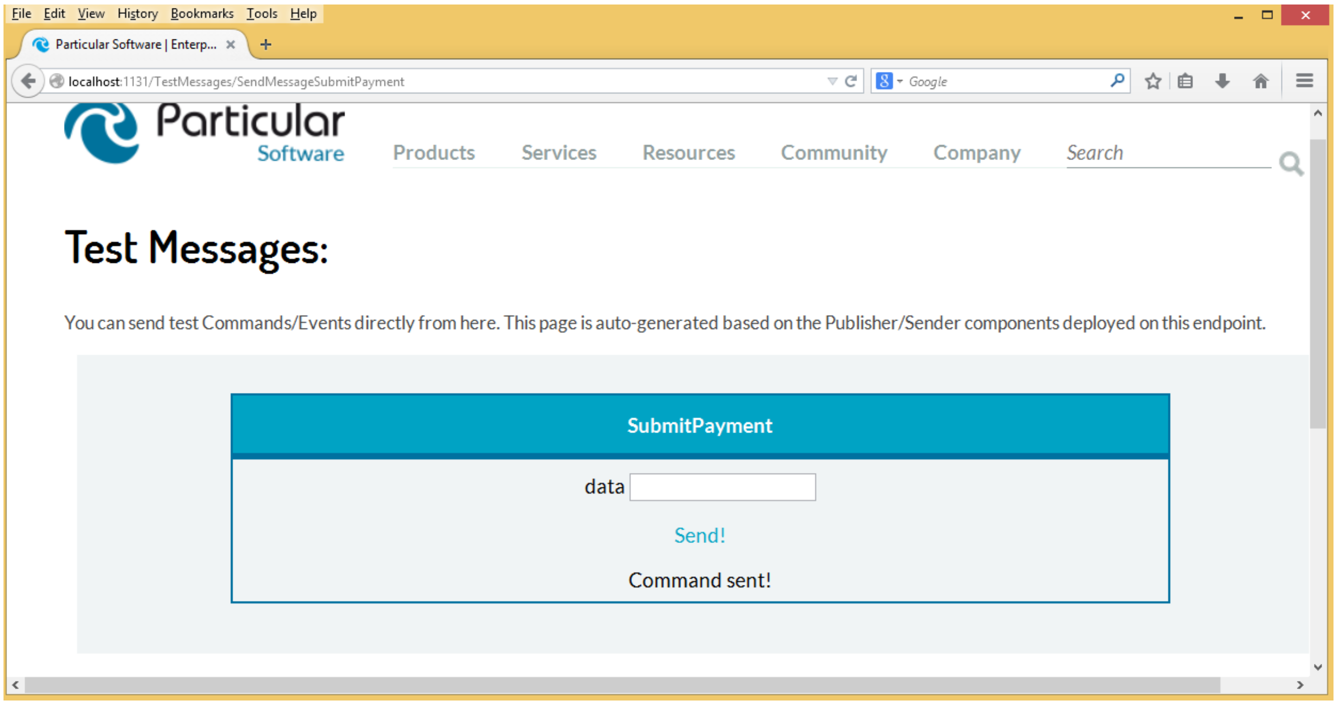
SubmitPayment

data

Send!

Command sent!

In this section, we will be using the **PaymentEngine-Saga** solution created with the ServiceMatrix, so ServiceMatrix must be installed. This will be the ServiceMatrix walkthrough. This will be a publish-subscribe and Saga solution built using ServiceMatirx. The outcome of a sent payment will look like::


File Edit View History Bookmarks Tools Help
Particular Software | Enterp...
localhost:1131/TestMessages/SendMessageSubmitPayment
Google
Particular Software
Products
Services
Resources
Community
Company
Search

Test Messages:

You can send test Commands/Events directly from here. This page is auto-generated based on the Publisher/Sender components deployed on this endpoint.

SubmitPayment

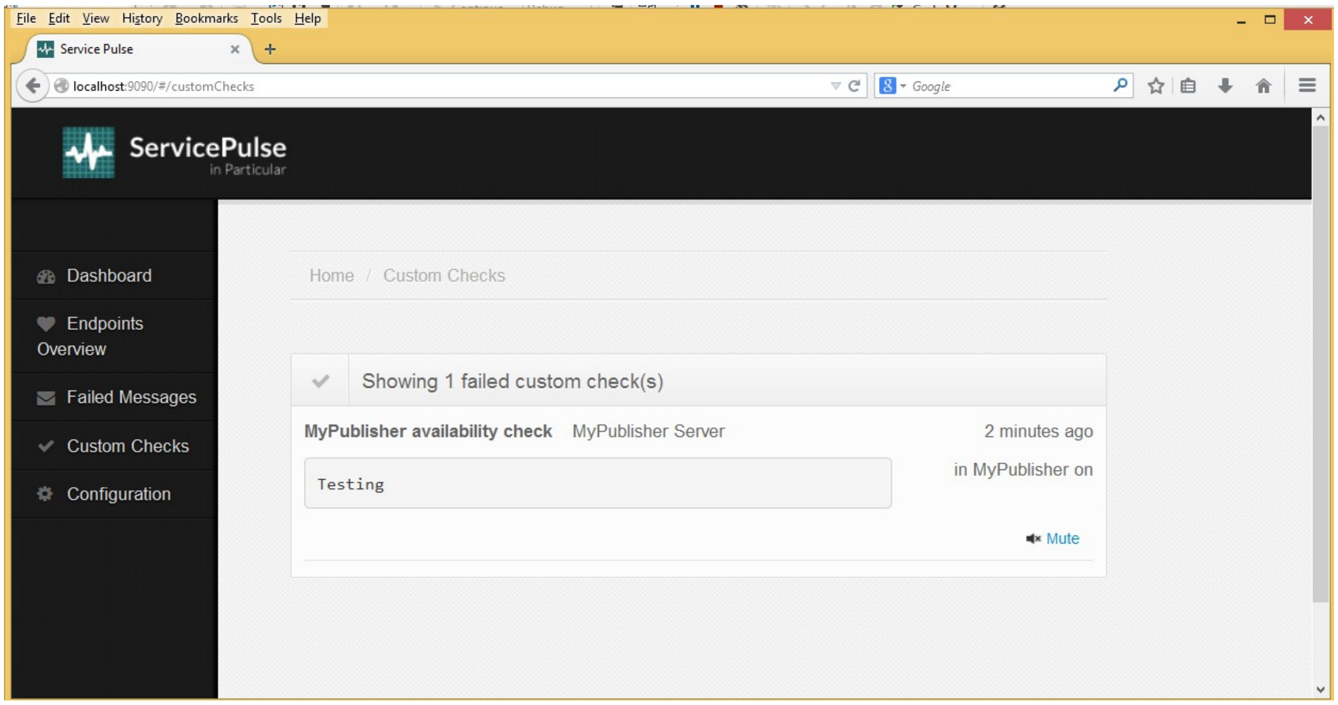
data

Send!

Command sent!

In this section, we will be using the **PubSub--ReportFailure** solution, the MyPublisher project reports a failure check that will be reported in the ServicePulse. This shows Custom Checks.

A failure check sent to ServicePulse will look like:



Showing 1 failed custom check(s)

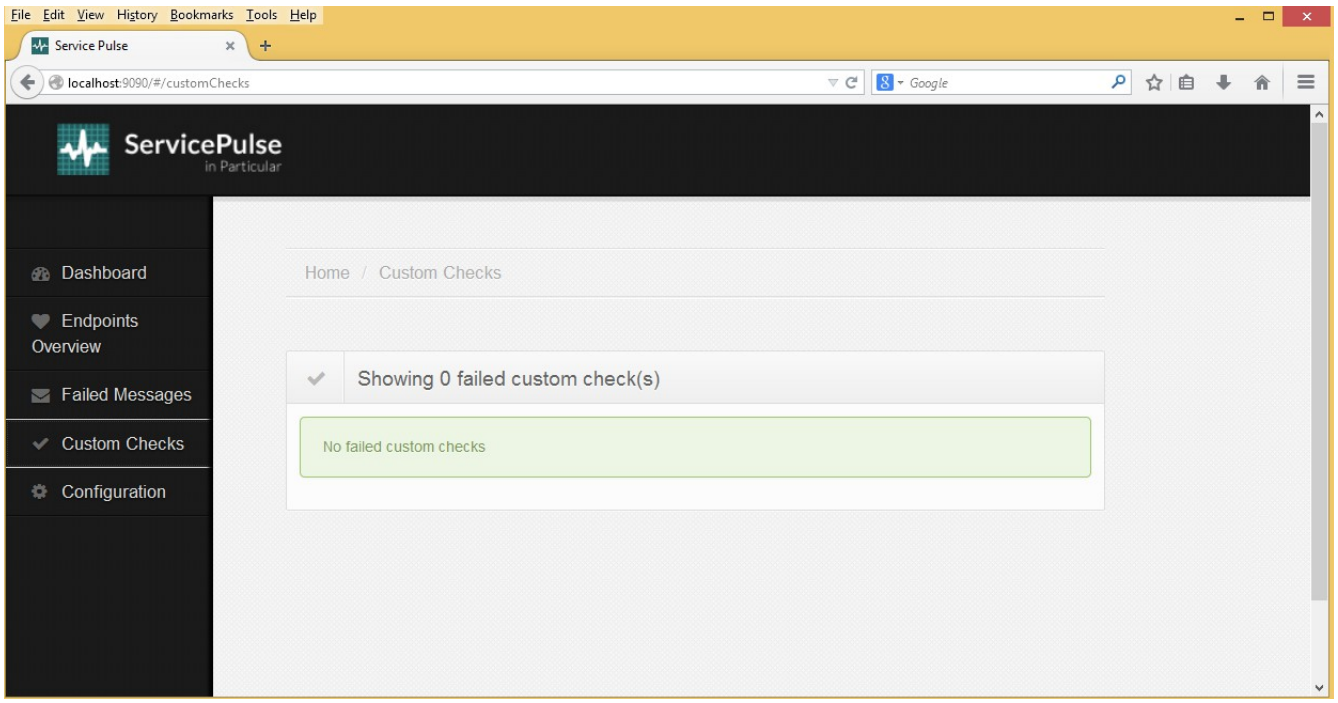
MyPublisher availability check MyPublisher Server 2 minutes ago

Testing

in MyPublisher on

In this section, we will be using the **PubSub--ReportPass** solution, the MyPublisher project reports a pass check that will be reported in the ServicePulse. This shows Custom Checks.

A report pass in ServicePulse will look like:



Showing 0 failed custom check(s)

No failed custom checks

We will be using the **ScaleOut-ServiceControl** solution. This solution is similar to an earlier chapter's example, except that there we added service control plugins through NuGet to generate service control endpoints for monitoring purposes.

In this section, we will be using the **ScaleOut** solution with the following projects:

- Orders.Messages – The common messages for the sender and handlers.
- Orders.Sender – Will send messages to Orders.Handler to be handled across the workers, worker1and worker2.
- Orders.Handler.Worker1 – One of the worker services that is using a worker profile to send a response back to the sender. This will be an additional worker copy of Orders.Handler.
- Orders.Handler.Worker2 – One of the worker services that is using a worker profile to send a response back to the sender. This will be an additional worker copy of Orders.Handler.
- Orders.Handler– an endpoint which processes the message and configured to the distributor. This will be the master profile that the sender will send the place order command to in the "orders.handler" MSMQ. In the Visual Studio 2012 debugger, the "NServiceBus.Integration NserviceBus.Master" is set in the command line to be used instead of "Configure.Instance.RunDistributor()".

You may need to run Visual Studio as an administrator.

- If one does not start up 'Orders.Handler' first and wait for it to be up-and-running the workers fail trying to access the distributor queue.